

UNIT32: OFFSCREEN RENDERING

【学習要項】

- ☐Frame buffer
- ☐Gaussian blur
- ☐Downsampling
- ☐Linear color space
- ☐HDR(High Dynamic Range)
- ☐SDR(Standard Dynamic Range)
- ☐Gamma correction
- ☐Inverse gamma correction(Degamma)
- ☐Tone mapping

【演習手順】

1. テスト用オブジェクトの生成と描画

- ①framework クラスの initialize メンバ関数で sprite_batch と skinned_mesh のオブジェクトを生成する

```
sprite_batches[0] = std::make_unique<sprite_batch>(device.Get(), L"¥¥resources¥¥screenshot.jpg", 1);
skinned_meshes[0] = std::make_unique<skinned_mesh>(device.Get(), ".¥¥resources¥¥nico.fbx");
```

- ②framework クラスの render メンバ関数で sprite_batch オブジェクトを背景として画面全体に描画する
※面カルリングなし、深度テストなし、深度ライトなし

- ③framework クラスの render メンバ関数で skinned_mesh オブジェクトを描画する
※面カルリングあり、深度テストあり、深度ライトあり

- ④実行し、シーン（背景とキャラクタ）が描画されていることを確認する

2. オフスクリーンレンダリングを行うために、framebuffer クラスを定義・実装する

- ①framebuffer クラスの定義（プロジェクトに framebuffer.h を新規追加する）

```
1: #include <d3d11.h>
2: #include <wrl.h>
3: #include <cstdint>
4:
5: class framebuffer
6: {
7: public:
8:     framebuffer(ID3D11Device *device, uint32_t width, uint32_t height);
9:     virtual ~framebuffer() = default;
10:
11:     Microsoft::WRL::ComPtr<ID3D11RenderTargetView> render_target_view;
12:     Microsoft::WRL::ComPtr<ID3D11DepthStencilView> depth_stencil_view;
13:     Microsoft::WRL::ComPtr<ID3D11ShaderResourceView> shader_resource_views[2];
14:     D3D11_VIEWPORT viewport;
15:
16:     void clear(ID3D11DeviceContext* immediate_context,
17:         float r = 0, float g = 0, float b = 0, float a = 1, float depth = 1);
18:     void activate(ID3D11DeviceContext* immediate_context);
19:     void deactivate(ID3D11DeviceContext* immediate_context);
20:
21: private:
22:     UINT viewport_count{ D3D11_VIEWPORT_AND_SCISSORRECT_OBJECT_COUNT_PER_PIPELINE };
23:     D3D11_VIEWPORT cached_viewports[D3D11_VIEWPORT_AND_SCISSORRECT_OBJECT_COUNT_PER_PIPELINE];
24:     Microsoft::WRL::ComPtr<ID3D11RenderTargetView> cached_render_target_view;
25:     Microsoft::WRL::ComPtr<ID3D11DepthStencilView> cached_depth_stencil_view;
26: };
```

- ②framebuffer クラスの実装（プロジェクトに framebuffer.cpp を新規追加する）

```
1: #include "framebuffer.h"
2: #include "misc.h"
3:
4: framebuffer::framebuffer(ID3D11Device* device, uint32_t width, uint32_t height)
5: {
6:     HRESULT hr{ S_OK };
7:
```

```
8:     Microsoft::WRL::ComPtr<ID3D11Texture2D> render_target_buffer;
9:     D3D11_TEXTURE2D_DESC texture2d_desc{};
10:    texture2d_desc.Width = width;
11:    texture2d_desc.Height = height;
12:    texture2d_desc.MipLevels = 1;
13:    texture2d_desc.ArraySize = 1;
14:    texture2d_desc.Format = DXGI_FORMAT_R16G16B16A16_FLOAT;
15:    texture2d_desc.SampleDesc.Count = 1;
16:    texture2d_desc.SampleDesc.Quality = 0;
17:    texture2d_desc.Usage = D3D11_USAGE_DEFAULT;
18:    texture2d_desc.BindFlags = D3D11_BIND_RENDER_TARGET | D3D11_BIND_SHADER_RESOURCE;
19:    texture2d_desc.CPUAccessFlags = 0;
20:    texture2d_desc.MiscFlags = 0;
21:    hr = device->CreateTexture2D(&texture2d_desc, 0, render_target_buffer.GetAddressOf());
22:    _ASSERT_EXPR(SUCCEEDED(hr), hr_trace(hr));
23:
24:    D3D11_RENDER_TARGET_VIEW_DESC render_target_view_desc{};
25:    render_target_view_desc.Format = texture2d_desc.Format;
26:    render_target_view_desc.ViewDimension = D3D11_RTV_DIMENSION_TEXTURE2D;
27:    hr = device->CreateRenderTargetView(render_target_buffer.Get(), &render_target_view_desc,
28:        render_target_view.GetAddressOf());
29:    _ASSERT_EXPR(SUCCEEDED(hr), hr_trace(hr));
30:
31:    D3D11_SHADER_RESOURCE_VIEW_DESC shader_resource_view_desc{};
32:    shader_resource_view_desc.Format = texture2d_desc.Format;
33:    shader_resource_view_desc.ViewDimension = D3D11_SRV_DIMENSION_TEXTURE2D;
34:    shader_resource_view_desc.Texture2D.MipLevels = 1;
35:    hr = device->CreateShaderResourceView(render_target_buffer.Get(), &shader_resource_view_desc,
36:        shader_resource_views[0].GetAddressOf());
37:    _ASSERT_EXPR(SUCCEEDED(hr), hr_trace(hr));
38:
39:    Microsoft::WRL::ComPtr<ID3D11Texture2D> depth_stencil_buffer;
40:    texture2d_desc.Format = DXGI_FORMAT_R24G8_TYPELESS;
41:    texture2d_desc.BindFlags = D3D11_BIND_DEPTH_STENCIL | D3D11_BIND_SHADER_RESOURCE;
42:    hr = device->CreateTexture2D(&texture2d_desc, 0, depth_stencil_buffer.GetAddressOf());
43:    _ASSERT_EXPR(SUCCEEDED(hr), hr_trace(hr));
44:
45:    D3D11_DEPTH_STENCIL_VIEW_DESC depth_stencil_view_desc{};
46:    depth_stencil_view_desc.Format = DXGI_FORMAT_D24_UNORM_S8_UINT;
47:    depth_stencil_view_desc.ViewDimension = D3D11_DSV_DIMENSION_TEXTURE2D;
48:    depth_stencil_view_desc.Flags = 0;
49:    hr = device->CreateDepthStencilView(depth_stencil_buffer.Get(), &depth_stencil_view_desc,
50:        depth_stencil_view.GetAddressOf());
51:    _ASSERT_EXPR(SUCCEEDED(hr), hr_trace(hr));
52:
53:    shader_resource_view_desc.Format = DXGI_FORMAT_R24_UNORM_X8_TYPELESS;
54:    shader_resource_view_desc.ViewDimension = D3D11_SRV_DIMENSION_TEXTURE2D;
55:    hr = device->CreateShaderResourceView(depth_stencil_buffer.Get(), &shader_resource_view_desc,
56:        shader_resource_views[1].GetAddressOf());
57:    _ASSERT_EXPR(SUCCEEDED(hr), hr_trace(hr));
58:
59:    viewport.Width = static_cast<float>(width);
60:    viewport.Height = static_cast<float>(height);
61:    viewport.MinDepth = 0.0f;
62:    viewport.MaxDepth = 1.0f;
63:    viewport.TopLeftX = 0.0f;
64:    viewport.TopLeftY = 0.0f;
65: }
66: void framebuffer::clear(ID3D11DeviceContext* immediate_context,
67:     float r, float g, float b, float a, float depth)
68: {
69:     float color[4]{ r, g, b, a };
70:     immediate_context->ClearRenderTargetView(render_target_view.Get(), color);
71:     immediate_context->ClearDepthStencilView(depth_stencil_view.Get(), D3D11_CLEAR_DEPTH, depth, 0);
72: }
```

```
73: void framebuffer::activate(ID3D11DeviceContext* immediate_context)
74: {
75:     viewport_count = D3D11_VIEWPORT_AND_SCISSORRECT_OBJECT_COUNT_PER_PIPELINE;
76:     immediate_context->RSGetViewports(&viewport_count, cached_viewports);
77:     immediate_context->OMGetRenderTargets(1, cached_render_target_view.ReleaseAndGetAddressOf(),
78:         cached_depth_stencil_view.ReleaseAndGetAddressOf());
79:
80:     immediate_context->RSSetViewports(1, &viewport);
81:     immediate_context->OMSetRenderTargets(1, render_target_view.GetAddressOf(),
82:         depth_stencil_view.Get());
83: }
84: void framebuffer::deactivate(ID3D11DeviceContext* immediate_context)
85: {
86:     immediate_context->RSSetViewports(viewport_count, cached_viewports);
87:     immediate_context->OMSetRenderTargets(1, cached_render_target_view.GetAddressOf(),
88:         cached_depth_stencil_view.Get());
89: }
```

3. オフスクリーンバッファ (framebuffer) に描画する

①framework クラスのメンバ変数を定義する

```
std::unique_ptr<framebuffer> framebuffers[8];
```

②framework クラスの initialize メンバ関数で framebuffer オブジェクトを生成する

```
framebuffers[0] = std::make_unique<framebuffer>(device.Get(), 1280, 720);
```

③framework クラスの render メンバ関数で描画先をオフスクリーンバッファ (framebuffer) に変更する

```
1: framebuffers[0]->clear(immediate_context.Get());
2: framebuffers[0]->activate(immediate_context.Get());
3:
4:     :
5:     : Calling the render functions of skinned_mesh and sprite_batch objects
6:     :
7:
8: framebuffers[0]->deactivate(immediate_context.Get());
```

④実行し、シーン (背景とキャラクタ) が描画されないことを確認する

4. framebuffer オブジェクトをテクスチャ (シェーダーリソースビュー) として扱い、画面に描画する

①fullscreen_quad_vs 頂点シェーダーの実装 (プロジェクトに fullscreen_quad_vs.hlsl を新規追加する)

```
1: #include "fullscreen_quad.hlsl"
2:
3: VS_OUT main(in uint vertexid : SV_VERTEXID)
4: {
5:     VS_OUT vout;
6:     const float2 position[4] = { { -1, +1 }, { +1, +1 }, { -1, -1 }, { +1, -1 } };
7:     const float2 texcoords[4] = { { 0, 0 }, { 1, 0 }, { 0, 1 }, { 1, 1 } };
8:     vout.position = float4(position[vertexid], 0, 1);
9:     vout.texcoord = texcoords[vertexid];
10:    return vout;
11: }
```

②fullscreen_quad_ps ピクセルシェーダーの実装 (プロジェクトに fullscreen_quad_ps.hlsl を新規追加する)

```
1: #include "fullscreen_quad.hlsl"
2:
3: #define POINT 0
4: #define LINEAR 1
5: #define ANISOTROPIC 2
6: SamplerState sampler_states[3] : register(s0);
7: Texture2D texture_map : register(t0);
```

```
8: float4 main(VS_OUT pin) : SV_TARGET
9: {
10:     return texture_map.Sample(sampler_states[LINEAR], pin.texcoord);
11: }
```

③fullscreen_quad クラスの定義（プロジェクトに fullscreen_quad.h を新規追加する）

```
1: #include <d3d11.h>
2: #include <wrl.h>
3: #include <cstdint>
4:
5: class fullscreen_quad
6: {
7: public:
8:     fullscreen_quad(ID3D11Device *device);
9:     virtual ~fullscreen_quad() = default;
10:
11: private:
12:     Microsoft::WRL::ComPtr<ID3D11VertexShader> embedded_vertex_shader;
13:     Microsoft::WRL::ComPtr<ID3D11PixelShader> embedded_pixel_shader;
14:
15: public:
16:     void blit(ID3D11DeviceContext *immediate_context, ID3D11ShaderResourceView** shader_resource_view,
17:         uint32_t start_slot, uint32_t num_views, ID3D11PixelShader* replaced_pixel_shader = nullptr);
18: };
```

④fullscreen_quad クラスの実装（プロジェクトに fullscreen_quad.cpp を新規追加する）

```
1: #include "fullscreen_quad.h"
2: #include "shader.h"
3: #include "misc.h"
4:
5: fullscreen_quad::fullscreen_quad(ID3D11Device *device)
6: {
7:     create_vs_from_cso(device, "fullscreen_quad_vs.cso", embedded_vertex_shader.ReleaseAndGetAddressOf(),
8:         nullptr, nullptr, 0);
9:     create_ps_from_cso(device, "fullscreen_quad_ps.cso", embedded_pixel_shader.ReleaseAndGetAddressOf());
10: }
11: void fullscreen_quad::blit(ID3D11DeviceContext *immediate_context,
12:     ID3D11ShaderResourceView** shader_resource_view, uint32_t start_slot, uint32_t num_views,
13:     ID3D11PixelShader* replaced_pixel_shader)
14: {
15:     immediate_context->IASetVertexBuffers(0, 0, nullptr, nullptr, nullptr);
16:     immediate_context->IASetPrimitiveTopology(D3D11_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP);
17:     immediate_context->IASetInputLayout(nullptr);
18:
19:     immediate_context->VSSetShader(embedded_vertex_shader.Get(), 0, 0);
20:     replaced_pixel_shader ? immediate_context->PSSetShader(replaced_pixel_shader, 0, 0) :
21:         immediate_context->PSSetShader(embedded_pixel_shader.Get(), 0, 0);
22:
23:     immediate_context->PSSetShaderResources(start_slot, num_views, shader_resource_view);
24:
25:     immediate_context->Draw(4, 0);
26: }
```

⑤framework クラスのメンバ変数を定義する

```
std::unique_ptr<fullscreen_quad> bit_block_transfer;
```

⑥framework クラスの initialize メンバ関数で fullscreen_quad オブジェクトを生成する

```
bit_block_transfer = std::make_unique<fullscreen_quad>(device.Get());
```

⑦framework クラスの render メンバ関数でオフスクリーンバッファ（framebuffer）の内容を画面に描画する ※面カルリングなし、深度テストなし、深度ライトなし

UNIT32: OFFSCREEN RENDERING

※動作確認後`#if-#endif` ディレクティブのコードは無効にすること(1:-4:行目)

```
1: #if 1
2:     bit_block_transfer->blit(immediate_context.Get(),
3:         framebuffers[0]->shader_resource_views[0].GetAddressOf(), 0, 1);
4: #endif
```

⑧実行し、シーン（背景とキャラクタ）が描画されていることを確認する

5. framebuffer オブジェクトに描画されたシーンから高輝度成分を抽出する

①framework クラスの initialize メンバ関数で framebuffer オブジェクトを生成する

```
framebuffers[1] = std::make_unique<framebuffer>(device.Get(), 1280 / 2, 720 / 2);
```

②luminance_extraction_ps ピクセルシェーダーの実装（プロジェクトに luminance_extraction_ps.hlsl を新規追加する）

```
1: #include "fullscreen_quad.hlsl"
2: #define POINT 0
3: #define LINEAR 1
4: #define ANISOTROPIC 2
5: SamplerState sampler_states[3] : register(s0);
6: Texture2D texture_maps[4] : register(t0);
7: float4 main(VS_OUT pin) : SV_TARGET
8: {
9:     float4 color = texture_maps[0].Sample(sampler_states[ANISOTROPIC], pin.texcoord);
10:    float alpha = color.a;
11:    color.rgb = smoothstep(0.6, 0.8, dot(color.rgb, float3(0.299, 0.587, 0.114))) * color.rgb;
12:    return float4(color.rgb, alpha);
13: }
```

③framework クラスのメンバ変数を定義する

```
Microsoft::WRL::ComPtr<ID3D11PixelShader> pixel_shaders[8];
```

④framework クラスの initialize メンバ関数でピクセルシェーダーオブジェクトを生成する

```
create_ps_from_cso(device.Get(), "luminance_extraction_ps.cso", pixel_shaders[0].GetAddressOf());
```

⑤framework クラスの render メンバ関数で framebuffers[0] から高輝度成分を抽出し framebuffers[1] に転送する
※動作確認後`#if-#endif` ディレクティブのコードは無効にすること(7:-10:行目)

```
1:     framebuffers[1]->clear(immediate_context.Get());
2:     framebuffers[1]->activate(immediate_context.Get());
3:     bit_block_transfer->blit(immediate_context.Get(),
4:         framebuffers[0]->shader_resource_views[0].GetAddressOf(), 0, 1, pixel_shaders[0].Get());
5:     framebuffers[1]->deactivate(immediate_context.Get());
6:
7: #if 1
8:     bit_block_transfer->blit(immediate_context.Get(),
9:         framebuffers[1]->shader_resource_views[0].GetAddressOf(), 0, 1);
10: #endif
```

⑧実行し、抽出された高輝度成分が描画されていることを確認する

※② 11: 行目の `smoothstep` のパラメータを変更し変化を確認する（定数バッファを定義して CPU から制御する）

6. 抽出された高輝度成分にブラーエフェクトをかける

①blur_ps ピクセルシェーダーの実装（プロジェクトに blur_ps.hlsl を新規追加する）

```
1: #include "fullscreen_quad.hlsl"
2: #define POINT 0
3: #define LINEAR 1
4: #define ANISOTROPIC 2
5: SamplerState sampler_states[3] : register(s0);
6: Texture2D texture_maps[4] : register(t0);
7: float4 main(VS_OUT pin) : SV_TARGET
```

```
8: {
9:     uint mip_level = 0, width, height, number_of_levels;
10:    texture_maps[1].GetDimensions(mip_level, width, height, number_of_levels);
11:
12:    float4 color = texture_maps[0].Sample(sampler_states[ANISOTROPIC], pin.texcoord);
13:    float alpha = color.a;
14:
15:    float3 blur_color = 0;
16:    float gaussian_kernel_total = 0;
17:
18:    const int gaussian_half_kernel_size = 3;
19:    const float gaussian_sigma = 1.0;
20:    [unroll]
21:    for (int x = -gaussian_half_kernel_size; x <= +gaussian_half_kernel_size; x += 1)
22:    {
23:        [unroll]
24:        for (int y = -gaussian_half_kernel_size; y <= +gaussian_half_kernel_size; y += 1)
25:        {
26:            float gaussian_kernel = exp(-(x * x + y * y) / (2.0 * gaussian_sigma * gaussian_sigma)) /
27:                (2 * 3.14159265358979 * gaussian_sigma * gaussian_sigma);
28:            blur_color += texture_maps[1].Sample(sampler_states[LINEAR], pin.texcoord +
29:                float2(x * 1.0 / width, y * 1.0 / height)).rgb * gaussian_kernel;
30:            gaussian_kernel_total += gaussian_kernel;
31:        }
32:    }
33:    blur_color /= gaussian_kernel_total;
34:    const float bloom_intensity = 1.0;
35:    color.rgb += blur_color * bloom_intensity;
36:
37:    return float4(color.rgb, alpha);
38: }
```

- ②framework クラスの initialize メンバ関数でピクセルシェーダーオブジェクトを生成する

```
create_ps_from_cso(device.Get(), " blur_ps.cso", pixel_shaders[1].GetAddressOf());
```

- ③framework クラスの render メンバ関数でシーン画像とブラーをかけた高輝度成分画像を合成し、画面に出力する

※シーン画像とは framebuffers[0]のシェーダーリソースビューのこと

※高輝度成分画像とは framebuffers[1]のシェーダーリソースビューのこと

```
1: ID3D11ShaderResourceView* shader_resource_views[2]
2:     { framebuffers[0]->shader_resource_views[0].Get(), framebuffers[1]->shader_resource_views[0].Get() };
3: bit_block_transfer->blit(immediate_context.Get(), shader_resource_views, 0, 2, pixel_shaders[1].Get());
```

- ④実行し、元画像にブルームがかかっていることを確認する

※① 18:行目の gaussian_half_kernel_size の値を変更し変化を確認する

※① 19:行目の gaussian_sigma の値を変更し変化を確認する (定数バッファを定義して CPU から制御する)

※① 34:行目の bloom_intensity の値を変更し変化を確認する (定数バッファを定義して CPU から制御する)

7. リニア色空間で処理を行う

- ①テクスチャからサンプリングした色情報に逆ガンマ補正を施す

- ②sprite_ps.hlsl を変更する

```
1: #include "sprite.hlsl"
2: Texture2D color_map : register(t0);
3: SamplerState point_sampler_state : register(s0);
4: SamplerState linear_sampler_state : register(s1);
5: SamplerState anisotropic_sampler_state : register(s2);
6:
7: float4 main(VS_OUT pin) : SV_TARGET
8: {
9:     float4 color = color_map.Sample(anisotropic_sampler_state, pin.texcoord);
10:    float alpha = color.a;
11:    #if 1
12:        // Inverse gamma process
```

```
13:     const float GAMMA = 2.2;
14:     color.rgb = pow(color.rgb, GAMMA);
15: #endif
16:     return float4(color.rgb, alpha) * pin.color;
17: }
```

③skinned_mesh_ps.hlsl を変更する

```
1: float4 main(VS_OUT pin) : SV_TARGET
2: {
3:     float4 color = texture_maps[0].Sample(sampler_states[ANISOTROPIC], pin.texcoord);
4:     float alpha = color.a;
* 5: #if 1
* 6:     // Inverse gamma process
* 7:     const float GAMMA = 2.2;
* 8:     color.rgb = pow(color.rgb, GAMMA);
* 9: #endif
10:
11:     :
12:     :
13:     :
14:
15:     return float4(diffuse + specular, alpha) * pin.color;
16: }
```

③ blur_ps.hlsl を変更し画面への最終出力値にガンマ補正とトーンマッピングを施す

※6. ① 35:行目に下記コードを挿入する

```
1: #if 1
2:     // Tone mapping : HDR -> SDR
3:     const float exposure = 1.2;
4:     color.rgb = 1 - exp(-color.rgb * exposure);
5: #endif
6:
7: #if 1
8:     // Gamma process
9:     const float GAMMA = 2.2;
10:    color.rgb = pow(color.rgb, 1.0 / GAMMA);
11: #endif
```

④実行し、実行結果を確認する

※③ 3:行目の `exposure` の値を変更し変化を確認する（定数バッファを定義して CPU から制御する）

8. 画面の上下左右の端にアーティファクトノイズがでる、原因を考察し解決しなさい

※サンプラーステートの `D3D11_TEXTURE_ADDRESS_WRAP` が原因、`D3D11_TEXTURE_ADDRESS_BORDER` に変更（`BorderColor` 値に注意せよ）

9. 実行時中 VisualStudio の出力ウィンドウにシェーダーの警告（D3D11 WARNING）がでる、原因を考察し解決しなさい

※フレームバッファは同時にレンダーターゲットビューおよびシェーダーリソースビューとしてシェーダーにバインドすることはできない

※下記コードを `framework` クラスの `render` メンバ関数の先頭に挿入する

```
1: ID3D11RenderTargetView* null_render_target_views[D3D11_SIMULTANEOUS_RENDER_TARGET_COUNT]{};
2: immediate_context->OMSetRenderTargets(_countof(null_render_target_views), null_render_target_views, 0);
3: ID3D11ShaderResourceView* null_shader_resource_views[D3D11_COMMONSHADER_INPUT_RESOURCE_SLOT_COUNT]{};
4: immediate_context->VSSetShaderResources(0, _countof(null_shader_resource_views), null_shader_resource_views);
5: immediate_context->PSSetShaderResources(0, _countof(null_shader_resource_views), null_shader_resource_views);
```

【評価項目】

- ☐ オフスクリーンレンダリング
- ☐ ブルームエフェクト
- ☐ トーンマッピング